Steve Tolvaj

CIS-3207

Project 2: Developing a Linux Shell

10/15/2021

Program Description:

The “myshell” program is designed to be a simpler Linux shell that can support similar features and commands. The first file named “myshell.c” is where the arguments found either in the file or from standard input are first parsed by spaces or tabs and formatted to remove any white space. A prompt is printed for the user if the input is from standard input. They are then checked for any redirection, piping, or background execution commands. While this occurs, they are also checked if the commands entered are either system commands or built-in commands (see readme_doc for basic usage instructions).

When the program starts two environment variables are set. There is a new environment variable initialized named “shell” (lowercase) that stores the current directory used to execute the myshell program (used later for readme_doc in the help function). The initial “PATH” variable is also cleared and is set to only include “/bin”.

Built-in commands are contained in the second file named “utility.c”. There are 8 different built-in commands: cd, clr, dir, environ, echo, help, pause, and quit. There are only four commands that support output redirection, which is: dir, environ, echo, and help. There is a header file used to contain the prototypes for the utility functions. The utility functions will be explained in greater detail below.

If any errors are encountered, they will be handled by printing the only error message as “An error has occurred”. If this error occurred while opening/reading the batch file, the program will exit. For all other errors, the program will continue and if the command is a system command, that system tool will handle the error. Many functions return -1 for error checking and messaging. If the child process contains an error, the message will be printed to STDERR from the child process.

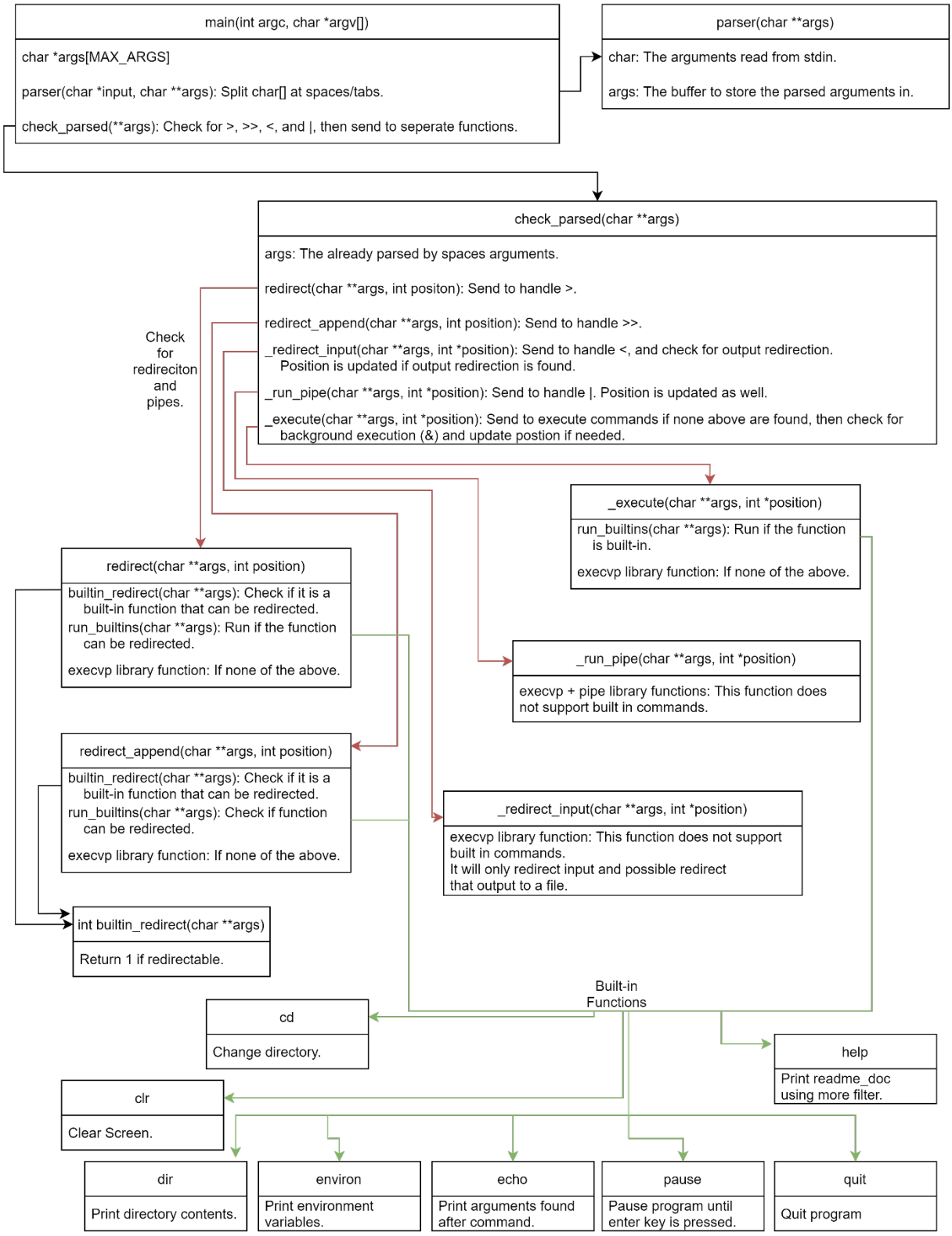
Program Design:

- **Myshell Functions**
    - The main function will first check if the arguments supplied during initial execution contain a batch file to read the commands from. If it does contain one, all commands will be executed line by line. If it does not contain a batch file a prompt is printed using the environment variable PWD followed by: myshell>. This is accomplished by using a flag to specify if the prompt should be printed and setting a FILE type to stdin or the batch file specified if available. All input is read line by line and then sent to a parser function. The arguments are stored in a 2D array of character arrays with a max length of arguments set to 100. This is local to the main method and is used to store the parsed commands.

- The parser function will split the line of commands by spaces or tabs and remove any extra white space. It will then be stored in the buffer supplied as an argument.
- The check_parsed function will search for any special characters (“<”, “>”, “>>”, “|”, or “&”) in the parsed arguments. If one is found it will set the first special character as NULL and send the arguments and the position the character was found to the correct function to run the commands. If no special characters were found it will run it as a normal child process.
- The redirect function will take the arguments with the position of the redirect character (not “>>” i.e., redirect append, handled in next function) after being set to null. It will then run the commands in a child process, up to the first null character. Then any output will be duplicated into the file descriptor that was supplied as a filename at the “>” position + 1. This function also supplies support for the built-in functions that work with output redirection. If this is the case, it will run in the parent process but use the same file descriptor at the position argument +1.
- The redirect_append function (“>>”) will provide the same functionality as the redirect function but the open file descriptor will not truncate the file, but only append to the file that exists.
- The redirect_input function (“<”) will also accept arguments with an integer position pointer of where the first input redirect character was found. The position pointer is used to update the index variable in the check_parsed function if there was also output redirection found after the input filename to update the search loop for special characters. The filename found after the “<” character will be used to duplicate standard input. If there is an output redirection character found after the input filename. The output will be sent to the filename found after with a similar method as the previous functions. All commands will be run in a child process and the parent process will wait for the child to finish.
- The execute function will be used to handle commands without redirection but can handle background process execution. It will accept the arguments, the position of where a background execution character was found (“&”), and a flag to designate if the parent should wait for the child process to finish. It will separate the arguments found after the “&” to run as a separate command after the first if there were any more commands found. It will then run the commands in the child process and if the wait flag was not true it will then not wait and try to run the next set of arguments by recursively calling itself. The position parameter is also updated to update the index for the search loop in the check_parsed function.
- The pipe function will be used when a “|” is found in the check_parsed function. It will separate the arguments found before and after the “|”. The output from the first set of arguments will be used as the input to the second set of arguments after being piped. Both sets of commands will run in separate child processes and the parent will then wait for both to complete execution.
- The builtin_redirect function is used as a check to find if the built-in is indeed a built-in function and if it can be output redirected. It simply checks if the command entered matches the correct function definition and returns an appropriate integer to be used in the functions above.

    - The run_builtins function is used to run the correct function when the first argument supplied matches the correct function. It will also print error messages if any of the functions' return value matches an error value.
- **Utility Functions (built-ins)** – all functions that can throw errors can return -1 to indicate an error or 0 for success.
    - The change_dir function will change the current directory along with the PWD environment variable. The first argument in the set must be cd and then followed by a directory to change to. If no directory is specified it will print the current directory.
    - The clear_prompt function will clear the screen. This is accomplished by using ANSI codes along with the printf function. This function will be called when “cls” is entered as a command.
    - The print_directory function will print the current directory if there are no other arguments found after the “dir” command. If there is a directory specified after the first argument it will print the contents of that directory
    - The print_envp will print the environment variables or any other null-terminated array of character arrays. The parameter can be supplied with the extern environ global variable to loop and print all environment variables. It is called when the “environ” command is entered.
    - The echo function will print any characters found after the initial echo argument. All words/char arrays will be only space-separated and taken from the same arguments. It is called when the “echo” command is entered.
    - The print_help function will print the readme_doc found in the initial directory where the shell was executed from. This directory is saved as shell=<executed directory>. This saves the location of the readme_doc when navigating to different directories from within the shell. It also prints the readme_doc to the screen by using the more filter tool from a child process. It is called when the “help” command is entered.
    - The pause_prompt function simply pauses the user prompt/myshell until the enter key is pressed. It is called when the “pause” command is entered.
    - The add_path function will add any other user-defined path environment variables or delete all path environment variables if no other arguments are entered after the command “path”. The path variables are concatenated with a “:” separating them and can accept multiple different space-separated paths to include after the command.
    - The count_args function is used as a utility for the built-in functions as some utility functions may need a count on how many arguments are entered in total.

Function
Call
Diagram



Implementation and Testing:

- Changes: that occurred in design are up to date with what is shown above. The original design found in pseudocode.txt did have some added changes and features.
    - Prompt: The original design had a separate prompt function that would continue printing the prompt until an exit command was entered. This proved to be a bad design and would be simpler just to print the prompt in the main function after each line is read from stdin if no batch file was specified.
    - Batch File: There was also a readBatch function that would specifically read the batch file if there was one specified (if argc == 2). It proved to be easier just to make a FILE data type and set it to either the batch file or stdin if no batch file was specified. This would also set a flag to true if there would need to be a prompt printed for user input.
    - Parser: The parser function in the pseudocode used strtok which contained extra code and did not check for tabs or remove white space. The updated parser function uses strtok_r and is simpler with an empty while loop that keeps splitting at spaces and tabs until a null is stored in the array of character arrays.
    - Check Commands for <, >, >>, |, and &: The function check_parsed is updated from the original. Instead of immediately creating two temporary arrays from the delimiting character(s) it instead holds the position of the character(s) and sets it to null. It may still be split and stored separately in separate functions that coincide with the special character(s) used. For readability and simplicity, each “<”, “>”, “>>”, “|”, and “&” has its separate functions instead of using an integer flag to specify what type was needed.
    - Redirection: All redirection functions are very similar to the initial pseudocode implementation.
    - Check for background execution: The original implementation had a check wait function that would return the position of the “&” for background execution. This was removed since it is now handled in the check_parsed function.
    - Checking for built-in commands: This function was also changed to return a specific integer if it can also be output redirected, if it is built-in, or if it is not.
    - Running built-in commands: A new function was created to run the built-in commands and to error check and error codes.
    - Piping: Original implementation of pipe function was adequate, and no issues arose.
    - Built-ins: The built-in functions were mostly correctly designed except for a few testing issues found below.
- Testing issues:
    - When the help function was called after changing the directory within the program the original readme_doc could not be found. This is where the “shell” environment variable that was saved was concatenated with the readme_doc name to specify the file path.
    - Background execution had unwanted behavior when using wait. The cause of this was using wait instead of waitpid, the parent process may have been waiting for the wrong child process during testing and the user prompt would not print until this error was corrected.
    - When using functions that had filenames specified for redirection the loop that searched for the special commands was not updated after running so it would run some commands twice or just the filename which cause some errors. The passing of the

memory location of the position and updating it to specify the end of the arguments helped solve this issue.
    - All other testing performed was handled by error checking as the program was implemented such as no input, wrong filenames, incorrect commands/flags, and different types of commands along with built-in function testing.
- Future Improvements
    - Combining separate redirection functions into one and changing file descriptors instead of using separate functions.
    - When using background execution or piping, the remaining arguments found after those commands could be sent back to the check_parsed function to allow for multiple background execution and piping (this may also apply to output redirection).